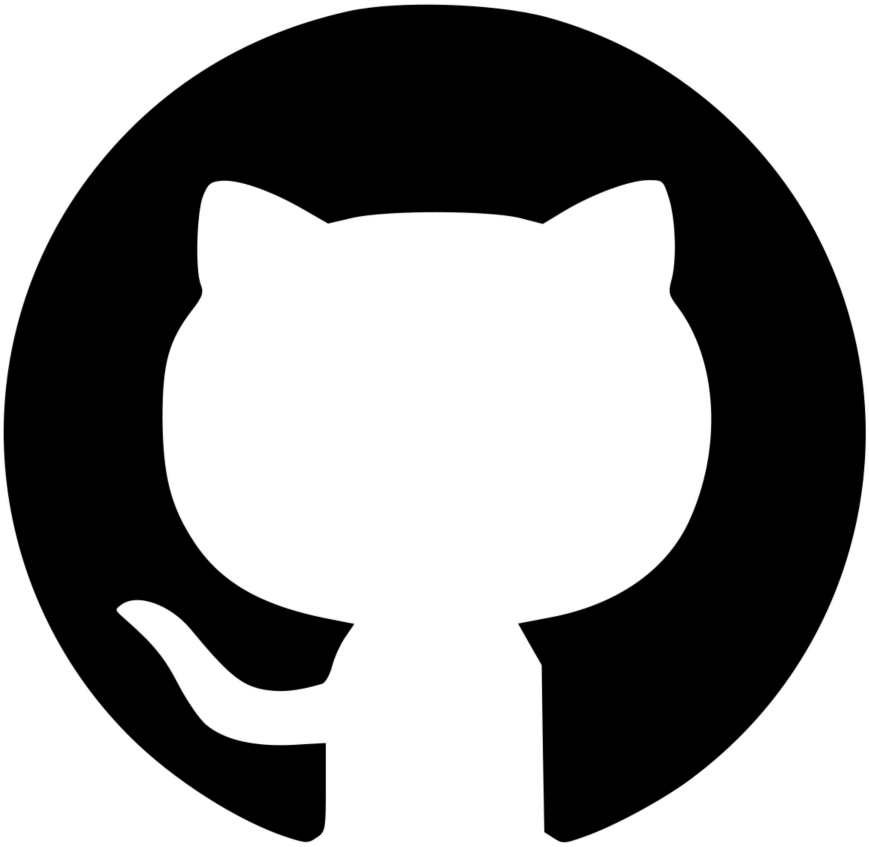
Matthias Lionardo 2B

SAE 2.02 Exploration Algorithmique



Sommaire

1. Efficacité

1.1 Méthodes Utilisées

Afin de trouver l'algorithme les plus efficaces, plusieurs techniques ont été mises en place :

- Tout d'abord un test supplémentaire pour mieux voir les performances des différents algos et analyser le temps d'exécution:

```
@Test
   public void testPerf(){
      try{
         assertEquals("SAE", (Algo.unRLE(Algo.RLE("SAE", 60), 60)));
      }
      catch(Exception e){
         fail("Exception inatendue");
      }
   }
```

  Tous les tests ont été effectués sur la même machine afin de ne pas fausser les résultats.
- J'ai également utilisé l'outil codacy.com qui permet de voir la complexité d'un algorithme.

Afin de mieux départager les algorithmes entre eux, chacun sera également évalué sur le barème pour noter les algorithmes et noter en fonction.

1.2 Codes

1.2.1 05efficacite.c

Cet algorithme a une vitesse d'exécution de 3 secondes avec le test de performance.

Sur codacy, le code est mal noté :

Cet algorithme obtient la note de 18/20 car il ne passe pas l'un de mes tests ajoutés :

Lorsqu'on demande à ce qu'il décompresse une chaîne de caractère qui l'est déjà, une erreur se produit.

1.2.2 44efficacite.java

Cet algorithme a une vitesse d'exécution de 4 secondes avec le test de performance.

Une complexité de 6 sur codacy.

Cet algorithme obtient la note de 18/20 car il ne passe pas l'un de mes tests ajoutés :

Lorsqu'on demande à ce qu'il décompresse une chaîne de caractère qui l'est déjà, une erreur se produit.

1.2.3 50efficacite.java

Cet algorithme ne passe pas le test de performance, à cause d'un problème de mémoire. Il a donc fallu baisser le nombre d'itérations dans le test performance pour pouvoir le classer si jamais un autre algorithme a également ce problème.
J'ai donc mis 40 pour le nombre d'itérations :

```
assertEquals("SAE", (Algo.unRLE(Algo.RLE("SAE", 40), 40)));
```

Il réalise ce nouveau test en environ 18 secondes.

B AlgoANoter / Efficacite / 50efficacite.java 4 5

Une complexité de 5 sur codacy.

Cet algorithme obtient la note de 18/20 car il ne passe pas l'un de mes tests ajoutés, le test de performances.

1.2.4 49efficacite.java

Cet algorithme aussi ne passait pas le test de performances et il a donc fallu également baisser les itérations à 40.

Il réalise ce test en environ 16.5 secondes.

Cet algorithme obtient la note de 18/20 car il ne passe pas l'un de mes tests ajoutés, le test de performances.

1.2.5 60efficacite.java

Cet algorithme aussi ne passait pas le test de performances et il a donc fallu également baisser les itérations à 40.

Il réalise ce test en environ 17 secondes.

B AlgoANoter / Efficacite / 66efficacite.java 3 5

Une complexité de 5 sur codacy.

Cet algorithme obtient la note de 18/20 car il ne passe pas l'un de mes tests ajoutés, le test de performances.

1.3 Classement

Place	Nom de l'algorithme
1	05efficacite.c
2	44efficacite.java
3	49efficacite.java
3	60efficacite.java
5	50efficacite.java

Le premier algorithme est le C car c'est le plus rapide, le C est souvent considéré comme le langage le plus performant ce qui se confirme ici.
Ensuite, on peut voir une grosse différence entre le deuxième algo et les trois derniers alors qu'ils sont tous les trois en java.
Cela s'explique principalement par l'utilisation de `'StringBuilder'` qui permet d'éviter la surcharge de mémoire et de temps due à la création répétée d'objets intermédiaires.

2. Simplicité

2.1 Méthodes Utilisées

Afin de trouver l'algorithme les plus simple, plusieurs techniques ont été mises en place :

- Tout d'abord, mon avis.
- J'ai également utilisé l'outil codacy.com qui permet de voir la qualité d'un algorithme.

Afin de mieux départager les algorithmes entre eux, chacun sera également évalué sur le barème pour noter les algorithmes et noté en fonction.

2.2 Codes

2.2.1 01simplicite.py

Cet algorithme est compliqué à comprendre même si il n'a pas beaucoup de lignes, il utilise des méthodes jamais vu. De plus, ce code nécessite au minimum python 3.12 afin de pouvoir l'utiliser.

Noté A sur codacy.

Cet algorithme obtient la note de 10/20 car il manque les deux méthodes récursives.

2.2.2 07simplicite.java

Cet algorithme est bien commenté et permet de bien le comprendre.

Noté B sur Codacy.

Cet algorithme obtient la note de 20/20, il a toutes les conditions attendues.

2.2.3 21simplicite.java

Cet algorithme est bien commenté mais trop, il y a 3/4 lignes de commentaires avant une ligne ce qui charge beaucoup le code.

Noté B sur codacy.

Cet algorithme obtient la note de 18/20 car il ne passe pas l'un de mes tests ajoutés :

Lorsqu'on demande à ce qu'il décompresse une chaîne de caractère qui l'est déjà, une erreur se produit.

2.2.4 51simplicite.py

Sur cet Algorithme il n'y a aucun commentaire ce qui ne facilite pas la compréhension.

Noté A sur codacy.
Cet algorithme obtient la note de 2/20. Il n'y a que la méthode RLE qui de plus ne passe pas tous les tests.

2.2.5 65simplicite.py

Sur cet algorithme il n'y a pas de commentaire, cependant les noms des variables sont assez explicites ce qui permet de mieux comprendre.

Noté A sur codacy.
Il obtient 20/20, il a toutes les conditions attendues.

2.3 Classement

Place	Nom de l'algorithme
1	07simplicite.java
2	65simplicite.py
3	21simplicite.java
4	01simplicite.py
5	51simplicite.py

Afin d'avoir un code simple il y a plusieurs choses auxquelles penser. D'abord mettre des noms de variables explicites et ensuite ne pas trop commenter l'ensemble.

3. Sobriété

3.1 Méthodes Utilisées

Afin de trouver l'algorithme les plus simple j'ai tout d'abord utiliser l'outil codacy.com qui permet de voir la qualité d'un algorithme.

3.2 Codes

3.2.1 05sobriete.c

Noté E sur codacy, le C n'est pas la meilleur option.

3.2.2 59sobriete.java

Noté A sur codacy, le C n'est pas la meilleur option.

3.2.3 62sobriete.java

Noté A sur codacy, le C n'est pas la meilleur option.

3.2.4 66sobriete.java

Noté A sur codacy, le C n'est pas la meilleur option.

3.3 Classement

Place	Nom de l'algorithme
1	66sobriete.java
1	62sobriete.java
1	59sobriete.java
4	05sobriete.c